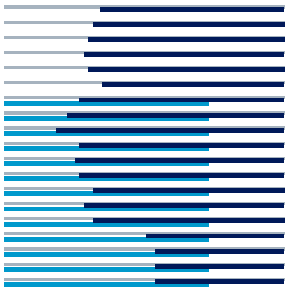
Co-Evolving Recurrent Neurons Learn Deep Memory POMDPs

Faustino J. Gomez
Jürgen Schmidhuber



Technical Report No. IDSIA-17-04
December 21, 2004

IDSIA / USI-SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
Galleria 2, 6928 Manno, Switzerland

Co-Evolving Recurrent Neurons Learn Deep Memory POMDPs

Faustino J. Gomez
Jürgen Schmidhuber

December 21, 2004

Abstract

Recurrent neural networks are theoretically capable of learning complex temporal sequences, but training them through gradient-descent is too slow and unstable for practical use in reinforcement learning environments. Neuroevolution, the evolution of artificial neural networks using genetic algorithms, can potentially solve real-world reinforcement learning tasks that require deep use of memory, i.e. memory spanning hundreds or thousands of inputs, by searching the space of recurrent neural networks directly. In this paper, we introduce a new neuroevolution algorithm called Hierarchical Enforced SubPopulations that simultaneously evolves networks at two levels of granularity: full networks and network components or *neurons*. We demonstrate the method in two POMDP tasks that involve temporal dependencies of up to thousands of time-steps, and show that it is faster and simpler than the current best conventional reinforcement learning system on these tasks.

1 Introduction

Neural networks with feedback connections or *recurrent* neural networks (RNNs) can potentially solve challenging sequential decision tasks where the correct choice of action at each time step can depend on the entire history of previous network inputs. This ability to act based upon the recollection of past events is essential for truly complex behavior. Unfortunately, training RNNs with standard gradient-descent methods such as Real-Time Recurrent Learning (RTRL; [1, 2]) or Back Propagation Through Time (BPTT; [3]) is not possible when actions depend on inputs from more than as few as 10 time-steps in the past. The error signal used to adjust synaptic weights either vanishes or explodes as it is propagated back through time, making learning prohibitively slow or preventing it altogether [4, 5]. For this reason, using conventional RNN architectures to approximate a value-function or policy for reinforcement learning (i.e. Q-learning, SARSA) is not practical when the environment exhibits long temporal dependencies.

Long Short-Term Memory (LSTM; [6]) overcomes this problem of unreliable gradient by using gated memory cells that guarantee constant error flow, and enable it to efficiently learn temporal dependencies spanning thousands of time-steps. Difficult POMDPs can be solved using LSTM as a value-function approximator for, e.g. Advantage Learning(λ) (RL-LSTM;[7]), but this approach is still slow due to the large data requirements of temporal difference learning.

Neuroevolution (NE; [8, 9, 10]), the evolution of neural networks using a genetic algorithm, circumvents both the problem of training recurrent networks and value-function approximation by searching the space of policies directly. Because NE uses only one learning component, a neural

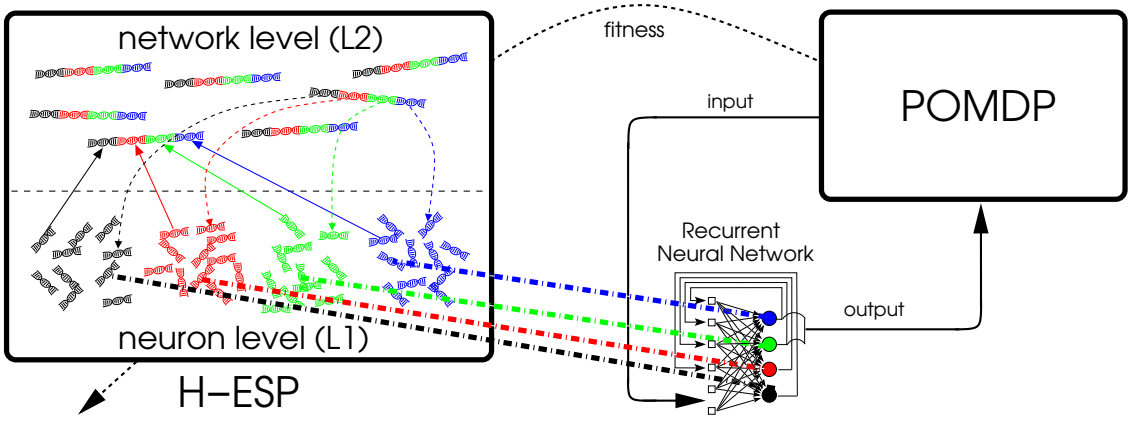


Figure 1: **H-ESP.** Evolution occurs at two levels, the neuron level (L1) and the network level (L2). L1 consists of multiple subpopulations of neurons, shown here in different colors. L2 consists of complete network representations that have either migrated up from L1 or have been created by recombining networks within L2. During evolution, networks are evaluated in two possible ways: from L2 directly, and from L1 by randomly selecting a neuron from each subpopulation and combining them into a complete network. The dashed lines from the neuron level to the network being evaluated indicate a network formed in this manner. A network from L2 that has higher fitness than any network formed so far in L1, has its neurons copied into their corresponding subpopulations in L1 (shown with the dashed arrows from L2 to L1). A network from in L1 that has higher fitness than the worst L2 network is copied into L2 (the solid arrows from L1 to L2). In this way, the two levels supply each other with new genetic material with which to search in their respective weight spaces.

network policy, it is conceptually simpler than value-function based methods, and is naturally suited to high-dimensional, continuous state and action spaces. However, up to now NE has only been applied to reinforcement learning tasks that require a very limited use of memory. For example, in partially observable environments where information from a small number of previous inputs is sufficient to determine the underlying process state.

In this paper, we demonstrate how neuroevolution can be used effectively for reinforcement learning problems that require *deep* use of memory. We introduce a new neuroevolution method, Hierarchical Enforced SubPopulations (H-ESP) that searches the space of recurrent networks by evolving at two levels in tandem: the level network components or *neurons*, and the level of full networks.

The next section describes the H-ESP algorithm. In section 3, we present the results of two experiments that show how neuroevolution can outperform even the current best single agent method on tasks with long term dependencies. In section 4 we analyze the behavior of the algorithm, and in sections 5 and 6, we discuss the results, and summarize our conclusions.

2 Hierarchical Enforced SubPopulations (H-ESP)

H-ESP is a new neuroevolution algorithm based on Enforced SubPopulations (ESP; [11]). Like ESP it evolves neural networks by coevolving network functional units or *neurons*. However, in addition to ESP's neuron level, H-ESP also evolves at the level of complete networks. The neuron level (L1) and network level (L2) are evolved simultaneously and interact by passing genetic material to each other in response to their relative performance (figure 1). H-ESP works as follows:

1. Initialization

   (a) **Neuron Level**. The number of hidden units u in the networks that will be evolved is specified and a subpopulation of n neuron chromosomes is created for each hidden unit. Each chromosome encodes the input, output, and recurrent connection weights of neuron with a string of random real numbers (figure 2).

   (b) **Network Level**. N random networks with u hidden units are formed.

2. Evaluation

   (a) **Neuron Level**. A neuron is selected at random from each of the u subpopulations, and combined to form a recurrent network. The network is evaluated on the task and awarded a fitness score which is assigned to each neuron in the network. If the fitness of the network is better than that of the worst network in L2 then it is inserted in L2. This process is repeated until each neuron has been used in a network.

   (b) **Network Level**. Evaluate the fitness of each network that has not already been evaluated. If a network has a higher fitness than the best network evaluated so far in L1, insert each of its neurons into the corresponding L1 subpopulation.

3. Recombination

   (a) **Neuron Level**. For each subpopulation the neurons are ranked by fitness, and the top quartile is recombined using 1-point crossover and mutation to create new neurons that replace the lowest-ranking half of the subpopulation.

   (b) **Network Level**. Each network is mated to a network with higher fitness using u-point crossover (i.e. one crossover point per neuron) and mutation to produce two new networks.

4. Repeat the Evaluation-Recombination cycle until a sufficiently fit network is found.

The neuron level (i.e. plain ESP) searches the space of networks indirectly by sampling the possible networks that can be constructed from the subpopulations of neurons. Network evaluations provide a fitness statistic with which to search for better neurons that can be eventually combined into a successful network. This cooperative coevolutionary approach [**?**] is an extension to Symbiotic, Adaptive Neuroevolution (SANE; [12]) which also evolves neurons, but in a single population. By using separate subpopulations, ESP accelerates the specialization of neurons into different sub-functions needed to form good networks because members of different evolving sub-function types are prevented from mating. Subpopulations also reduce noise in the neuron fitness measure because each evolving neuron type is guaranteed to be represented in every network that is formed. This allows ESP to evolve recurrent networks, where SANE could not.

Adding the network level is motivated by the observation that ESP discards each network as soon as it is evaluated, regardless of its fitness. This means that information about potentially fruitful neuron *combinations* is lost, especially in the early stages of evolution when diversity is high and re-sampling similar neuron combinations is unlikely. By maintaining a repository or "hall of fame" of the best networks, ESP can search within the space of highly fit neuron combinations in a way that is not possible at the neuron level because it constructs networks at random. The hope is that the L2 will make ESP less susceptible to premature convergence by preserving useful networks, and help fine-tune solutions.

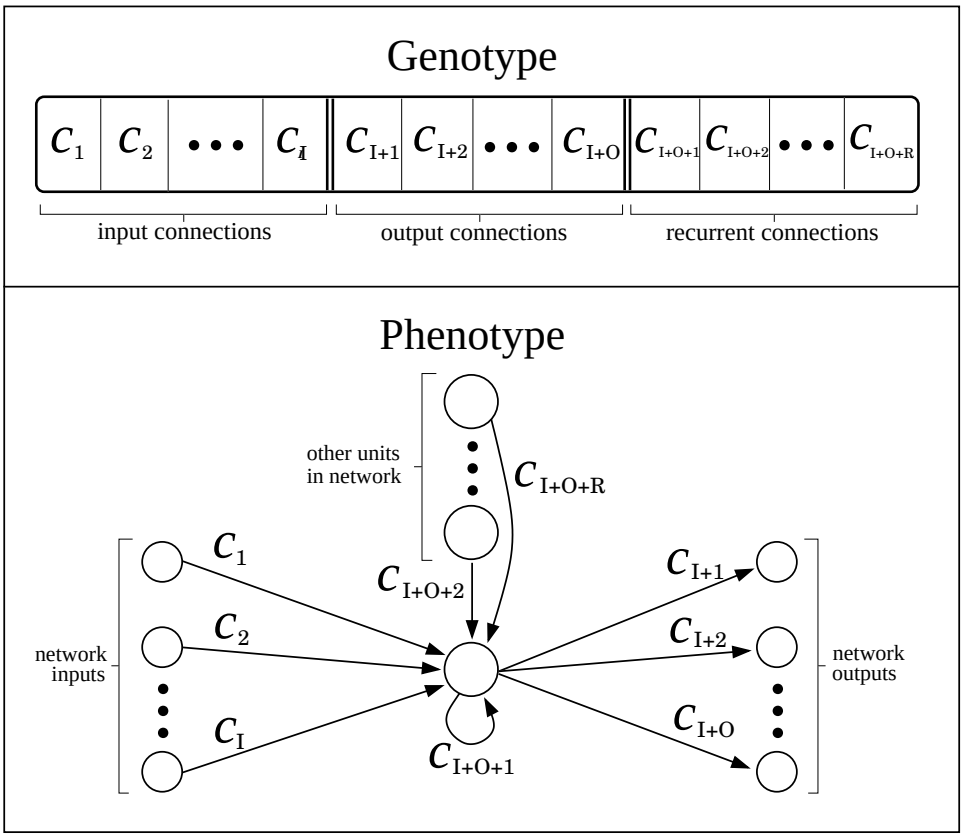


Figure 2: **Neuron genotype encoding.** The neuron-level genotypes encode the synaptic connection weights c_i of a single neuron as real numbers. The figure illustrates the mapping from genotype (top) to phenotype (bottom). Each chromosome has I input connections, O output connections, and R connections from the other neurons in the network, where R is equal to the number of subpopulations (network size), u. The network-level genotypes consist of a concatenation of u neuron genotypes.

If the performance of H-ESP does not improve for a predetermined number of generations, a technique called *burst mutation* is used. The idea of burst mutation is to search the space of modifications to the best solution found so far. When burst mutation is activated, the best neuron in each subpopulation is saved, the other neurons are deleted, and new neurons are created for each subpopulation by adding Cauchy distributed noise to its saved neuron. Evolution then resumes, but now searching in a neighborhood around the previous best solution. Burst mutation injects new diversity into the subpopulations and allows ESP to continue evolving after the initial subpopulations have converged.

3 Experiments

In this section, we evaluate H-ESP in two POMDP reinforcement learning tasks that require learning temporal dependencies of up to thousands of time steps: the T-maze, and the two-mode pole balancer. Because RL-LSTM is, to our knowledge, currently the most efficient method for solving this class of problems, our experiments use the same setups and compare our results with H-ESP to those published in [7]. To quantify the advantage of evolving hierarchically we also compare with normal ESP.

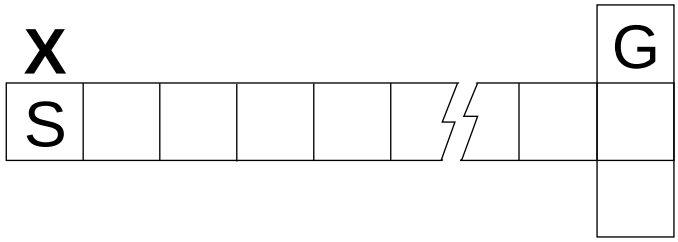


Figure 3: **Long term dependency T-maze.** Starting in S, the agent must traverse the corridor while remembering the signal X indicating the location of the goal G at the T-junction.

3.1 Long term dependency T-maze

The T-maze task is a discrete non-Markov problem designed to test a recurrent network's ability to learn long term dependencies. The maze consists of a corridor of n rooms with a start state S at one end and a T-junction at the opposite end (figure 3). Starting in S, the objective is to travel down to the end of the corridor and go either north or south at the T-junction depending on a signal X received in S that indicates the location of the goal G. In order to chose the correct direction the network must remember X for at least n time-steps.

3.1.1 Setup

The control agent can make 4 possible observations in the T-maze: if it is in S, it sees either 011 or 110 depending on whether G is to the north or south, respectively. If it is in the corridor it sees 101, and in the T-junction it sees 010. The agent has four possible actions: north, south, east, and west. The network receives a score of 4.0 for getting to the goal and -0.1 if it turns in the wrong direction at the T-junction or tries to go north or south in the corridor.

We ran 7 sets of simulations with different corridor lengths for both ESP and H-ESP, starting with length 10 and increasing to 70, in 10 room increments. Each set consisted of 10 runs evolving Elman networks with 3 input units (one for each input bit), 10 hidden units, and one output unit for each of the 4 possible actions. For both neuroevolution methods the subpopulation size was 200. For H-ESP the network level size was 100. Actions were selected probabilistically, weighted by the relative activation strength of each output unit. Each network was evaluated in 20 trials, 10 with the goal in the north room and 10 with the goal in the south room. The fitness was the lesser of the average scores from the two sets of 10 trials. After each generation the best network was re-tested using greedy actions, and the task was considered accomplished if this network achieved a perfect score of 4.0. The mutation rate was set to 0.8 for both methods, which means that each offspring has an 80% chance of having one of the weights in its chromosome replaced by a random value from the initial weight range, $[-10.0, 10.0]$. Burst mutation was activated if the best fitness found did not improve after 10 generations.

The neuroevolution methods were compared to four other methods, all using Advantage Learning but with a different choice of value-function representation: Long Short-Term Memory (RL-LSTM), Elman network with standard backpropagation (Elman-BP), Elman network with backpropagation through time (Elman-BPTT), and a table-based method (Memory Bits; [13]). As with H-ESP, the LSTM and the Elman networks used 3 input units and 4 output units, but the outputs produce action *values* and each output unit has a dedicated hidden layer. RL-LSTM used 12 standard hidden units and 3 memory cells. The Elman networks used 16 hidden units. The Memory Bits method uses a tabular representation with as many entries as there are observations times the number of memory bits. The control agent has additional actions that can set or unset

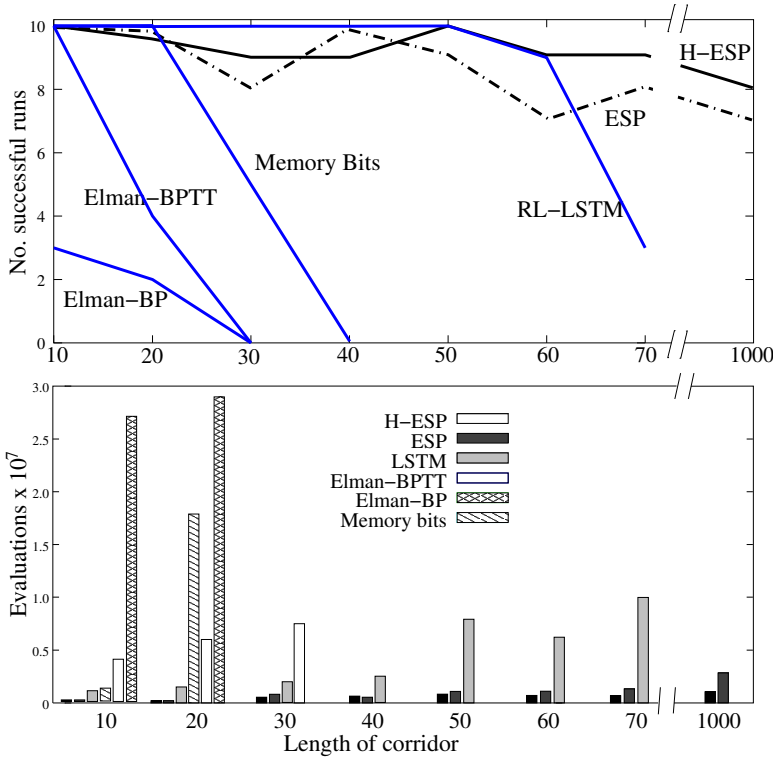


Figure 4: **T-maze results.** The upper plot shows the number of successful simulations out of 10 for each method compared. The lower plot shows the number of T-maze attempts required on average by each methods, for each corridor length. Datapoints for methods other than ESP and H-ESP taken from [7].

bits in order to disambiguate observations. For this task the agent has 1 bit since it only needs to remember one event.

3.1.2 Results

Figure 4 shows, for each corridor length and method, the number of maze navigation attempts required on average to solve the T-maze task, and the number of runs out of 7 that were successful. Above length 40 the only methods capable of solving the task are RL-LSTM, ESP, and H-ESP, with the two neuroevolution methods outperforming RL-LSTM by a factor of 10 on the hardest task of length 70. Importantly, the number of evaluations for H-ESP does not grow noticeably in relation to corridor length. In fact, even for length 1000 (far right in the figure), ESP and H-ESP were able to solve the task with a number of evaluations comparable to that required by LSTM a for corridor of length 20. The reason for this is that neuroevolution does not have to propagate the reinforcement signal back through time (i.e. down the corridor) to learn the task, and therefore the difficulty is less correlated to corridor length.

Up to length 40 the differences between ESP and H-ESP are not significant. The reason for this, we believe, is that in this task, these two algorithms either solve the task relatively quickly (in under 50 generations) or get stuck early on if there are no individuals in the initial population that can find either of the two goals, or are not heavily biased towards one of them. In these less common cases, some of which we have considered failure (upper plot in figure 4), the fine-tuning provided by the network level is not much help because a successful solution is too far away. For

Figure 5: **Two-mode pole balancing.** (a) The basic pole balancing system consists of a pole hinged to a cart that rolls in a finite stretch of track. (b) The controller representation for H-ESP is an Elman or Simple Recurrent Network (SRN) that receives the cart position, pole angle, and mode input at each time-step. The input is propagated through the hidden neurons which also receive the previous activation of the hidden layer via feedback (recurrent) connections, and then to the output unit which specifies the direction in which to apply the force to the cart.

the majority of cases, where the problem is solved quickly, the network level provides improved search efficiency that becomes more appreciable as the corridor gets longer and the weights may need more precise values.

Further testing showed that all of the networks evolved in corridors longer than length 20 where able to generalize to corridors of length 1 million.

3.2 Two mode pole balancing

The pole balancing problem has long been a standard benchmark for artificial learning systems. The basic system consists of a pole hinged to a wheeled cart on a finite stretch of track where the objective is to apply a force to the cart, at regular intervals, such that the pole is balanced indefinitely, and the cart stays within the track boundaries (figure 5a). In principle, pole balancing is a good test-bed because it has a continuous state space and requires learning from a delayed reward. However, in its classic setup, the task is simply too easy for this purpose as solutions can be found quickly through random search.

To make it more challenging, the basic task has been extended in various ways such as providing the controller with only the position of the cart, x, and the angle of the pole, θ, and not their respective velocities $(\dot{x}, \dot{\theta})$, as is normally the case [14]. To solve this non-Markov version, the control agent must utilize short-term memory to compute the velocities and determine the underlying state of the system. However, because only the current and most previous observations are necessary to compute the velocities (i.e. a one-step temporal dependency), this task requires minimal memory depth.

In [7], a version of the single pole task is introduced that involves temporal dependencies of thousands of time-steps. The controller agent must learn to operate in two modes: in Mode A, a positive control signal applies a 10 Newton force to the cart in the "right" direction, and a negative signal applies the force in the "left" direction; in Mode B the meaning of the control signals is reversed. In addition to only receiving x and θ, for the first 50 time-steps (or 1 second of simulated time) of each balancing attempt, the controller receives a "mode" signal through a special input unit that indicates which mode should be used. After the first second, the mode input is set to zero and the network must remember what mode to use to balance the pole for 100,000 time-steps.

The difficulty of this task is threefold: (1) the network must be able to balance the pole using

partial state information, (2) remember which mode it is in for thousands of time-steps, and (3) essentially implement a hybrid controller within a single architecture. That is, a controller that can operate discretely at a high level to switch between modes, and continuously at a lower level to balance the poles in each mode.

3.2.1 Setup

Networks were evaluated in one trial for each of the two modes. Each trial begins with the cart at the center of the track with the pole leaning 1 degree from vertical. At time $t = 0$ the controller starts to balance the pole applying a force to the cart every 0.02 second time-step until either the pole angle exceeds ± 12 degrees or the cart goes off the 4.8 meter track. The fitness was the number of time-steps in the shorter of the two trials. As with the T-maze task, this *maximin* fitness function prevents networks from being awarded high fitness for performing well in only one mode. In Mode A the mode input unit was set to 1.0, in Mode B it was set to -1.0.

For this task, Elman-BP, Elman-BPTT, Memory Bits, were not able to solve the task, so we compared H-ESP and ESP to RL-LSTM as it is the only method we know that has successfully solved this task. All three methods used networks with the following three inputs: the cart position, the pole angle, and the mode signal. RL-LSTM used LSTM networks with 14 standard hidden units, 6 memory cells, and two outputs, one for each action value. ESP and H-ESP evolved Elman networks with 5 units (figure 5b), a subpopulation size of 200, and, for H-ESP, a network level size of 100. The mutation rate, initial random weight range, and burst mutation criteria were the same as for the T-maze task.

3.2.2 Results

RL-LSTM was able to solve the task completely in 2 out of 10 simulations requiring and average of 6.25 million balancing attempts to do so. ESP was successful in 6 out of 10 runs with an average of 190,000 attempts, and H-ESP was successful in 8 of the 10 runs with an average of 121,000 attempts. In this continuous control task, H-ESP has a marked advantage over ESP. It seems that because this task is much more difficult than the T-maze task the network level has more to contribute by finding refinements to existing neuron combinations that the neuron level has missed.

One advantage that neuroevolution has on this task is that learning the actual balancing of the pole is trivial for evolutionary search, whereas value-function methods typically take much longer to learn just the balancing policy. Therefore, the challenge for H-ESP is to discover a network that can implement two policies simultaneously and remember the mode input for thousands of time steps.

Figure 6 shows the behavior of one of the evolved solutions operating in each of the control modes. The two top plots show the angle of the pole for each mode for the first 500 time-steps (10 seconds of simulated time). After 50 time-steps the mode signal is switched off causing an abrupt change in the network activation (shown for each unit in the lower plots). What is interesting about this solution is that it implements two independent strategies within such a small network. Some of the neurons go from a saturated steady state to more dynamic behavior (neuron 2) or vice-versa (neuron 5) when modes are switched.

4 Analysis of Network Level Contribution

Here we try to give some insight into how H-ESP uses the network level to improve search efficiency. Figure 7 shows the performance for a typical H-ESP run, and illustrates how the two evolutionary levels interact. The uppermost graph shows for each generation the fitness of the best network

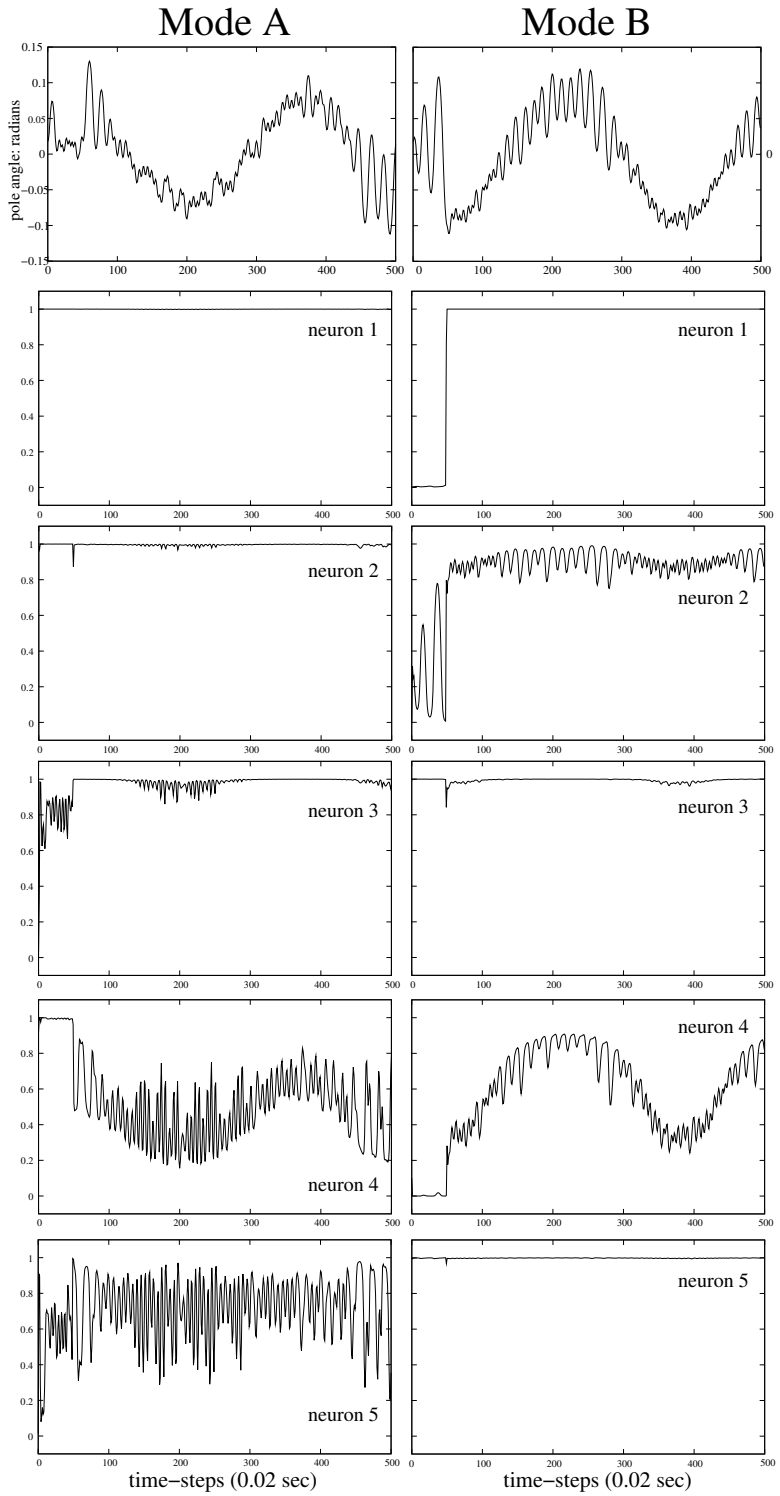


Figure 6: **Two-mode pole balancer behavior.** The plots show the behavior one of the evolved pole balancing controllers and its internal activation for the two modes of operation. The plots in the left column are for Mode A, on the right for mode B. The uppermost row shows the angle of the pole in radians. The rows below show the activation level of each of the 5 hidden units. It is clear that the network employs a different strategy in each mode. Likewise, the activation patterns in each mode are quite different.

found in that generation in L1, the highest fitness found so far within the current burst phase (i.e. from one burst mutation to the next) in L1, and the best fitness found so far in the entire system (L1 and L2). The five graphs below show the percentage of neurons in each L1 subpopulation that have at least one weight value that originated in the L2.

Initially, the two levels communicate in only one direction, from L1 to L2: networks evaluated in L1 move up to L2 where they evolve independently. Eventually, L2 discovers a network with higher fitness, and its neurons are percolated down to L1. The points at which this occurred in this example are indicated by the small arrows. These new neurons are then evaluated as part of some network in L1, and are either eliminated from their subpopulation, as is the case with the network inserted at generation 32, or they may be preserved because they contribute to a highly fit network. In this latter case, a new neuron may be selected for recombination and have its genetic material transmitted to future generations as occurs with the insertion at generation 68. Note that since the subpopulations evolve separately, the neurons derived from the same L2 network can have very different fates in L1, as can be is seen in the differences between the subpopulation plots.

The abrupt drop in the curves of subpopulations 1 and 2 near generation 120 is due to burst mutation which replaces the neurons in each subpopulation with mutated copies of the best neuron. The onset of burst mutation is indicated in the upper plot by the sharp downward spikes in the "burst phase best" curve.

Around generation 130, the network level jumps far ahead of the neuron level (the large step in the "overall best" curve), and material from L2 starts to spread throughout L1 as a succession of networks percolate down. The neuron level then starts to catch up, but not before a solution is found in L2 at generation 158. Note that the incorporation of L2 material into L1 is initiated by different L2 networks for different subpopulations. For instance, subpopulation 1 starts to take in L2 neurons at generation 126, whereas subpopulation 4 rejects this neuron by not selecting it for recombination, but does accept the neuron at generation 135. However, even if L2 networks do not get incorporated into L1 they can still be useful by providing a more highly fit solution with which to burst mutate.

The network level, as demonstrated by this example, seems to be most beneficial in the later stages of evolution when fine local search is more profitable.

5 Discussion

The results show that neural networks can be used to solve deep memory POMDPs without using a specialized learning architecture like LSTM. This is the first study that has solved reinforcement learning tasks with such long term temporal dependencies using evolutionary neural networks. LSTM is the state of the art RNN for supervised temporal sequence learning, so it is not surprising that it outperformed the other gradient descent methods. However, in an RL context, even LSTM can be slow because it is essentially being asked to learn from a non-stationary training set. The sequence of input patterns and targets change over time as the network itself is updated, and, therefore, the gradient information is noisy and only as good as the current training set.

Neuroevolution works in a fundamentally different way. By doing away with the value-function, it is a simpler approach: no modification to the networks is made during interaction with the environment so that an error gradient is not needed. This means that less complex network solutions can be found that have fewer and more simple processing units. Simpler solutions are desirable especially in engineering applications where end-users want controllers they can understand.

Although H-ESP represents an improvement over ESP, we hope that this initial effort will inspire more research into methods that concurrently search solution space at multiple levels of

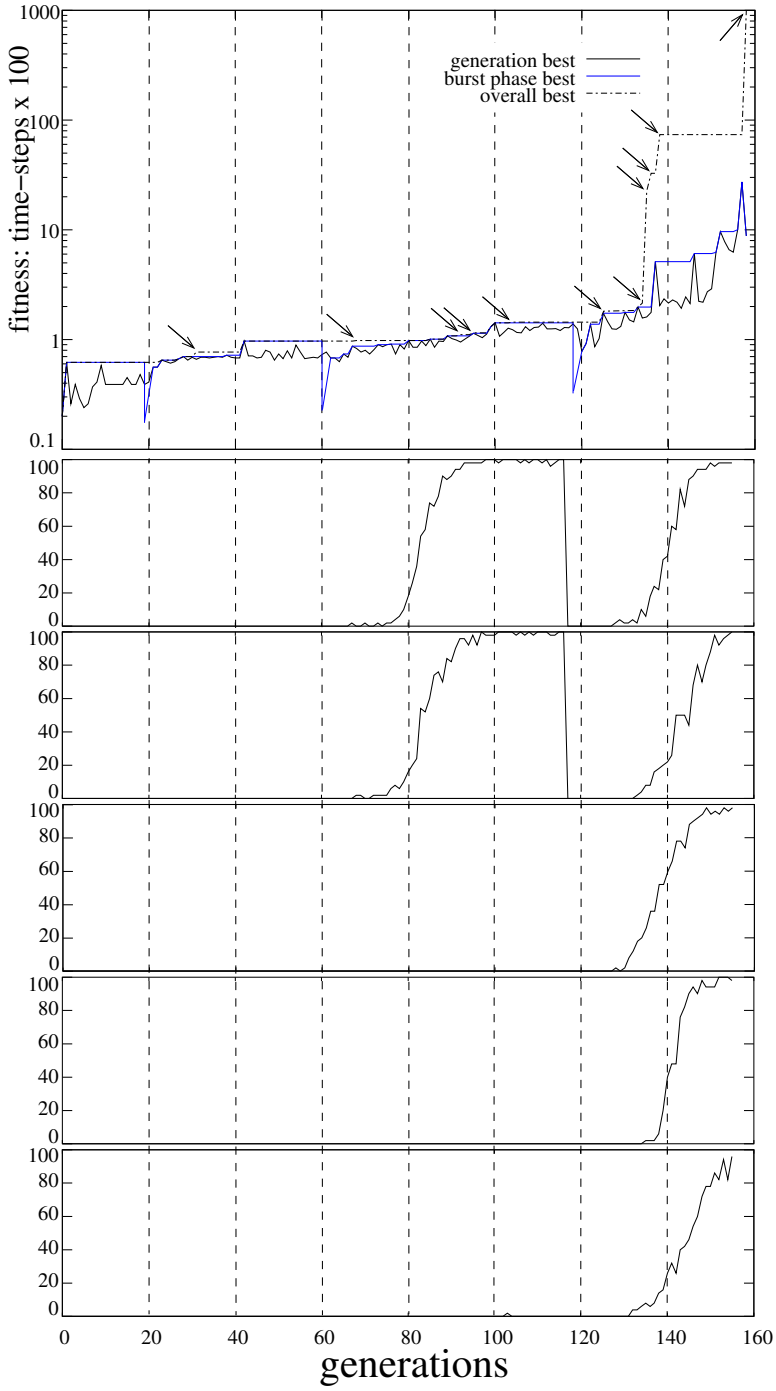


Figure 7: **H-ESP evolving a two-mode pole balancer.** Each of the curves in the uppermost graph (log-scale) represents a different fitness measure for a single run of H-ESP on the two-mode pole balancing task. The dark wavy curve is the fitness of best network from each generation. The gray curve is the best fitness so far for each burst mutation phase where the onset of each burst mutation is indicated by a downward spike in the curve. The dashed curve is the best overall fitness so far. Whenever L2 discovers a new overall best network, the dashed curve jumps above the gray curve at points indicated by the small diagonal arrows. The five graphs below show, for each L1 subpopulation, the percentage of neurons that have weight values originating in L2.

dimensionality. Our future work with H-ESP will first seek to analyze the inter-level dynamics in further detail. The current mechanism for communicating between the levels is very simple and ignores potentially useful information about what is happening inside each level. A more powerful approach might be to use diagnostics such as diversity to regulate the flow between levels in a more adaptive manner.

One possible extension to H-ESP is to add yet another level above the network level which contains modular networks composed of networks from L2. This third level (L3) would then communicate with L2, and indirectly with L1, in a fashion similar to the way the two levels do currently in H-ESP. This extension could be useful in tasks with high dimensional action space, such as anthropomorphic robot control, where the search space for bipedal walking strategies may be reduced if crosstalk between various control components (e.g. arms and legs) is limited by using separate, dedicated modules.

6 Conclusion

This paper has presented for the first time the successful application of a non gradient-descent neural network method to solving long term dependency POMDPs. We have introduced a new neuroevolution algorithm Hierarchical Enforced SubPopulations that enhances the already powerful Enforced SubPopulations by allowing it to search in the space of full network solutions as well as at the normal neuron level. This network level improves search efficiency, especially in the later stages of evolution by providing better fine-tuning of candidate solutions.

References

[1] Robinson, A.J., Fallside, F.: The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK (1987)

[2] Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation **1** (1989) 270–280

[3] Werbos, P.: Backpropagation through time: what does it do and how to do it. In: Proceedings of IEEE. Volume 78. (1990) 1550–1560

[4] Hochreiter, S.: Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München (1991)

[5] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **5** (1994) 157–166

[6] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9** (1997) 1735–1780

[7] Bakker, B.: Reinforcement learning with long short-term memory. In Dietterich, T., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14. Volume 14., Cambridge, MA, MIT Press (2002) 1475–1482

[8] Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87** (1999) 1423–1447

[9] Beer, R.D., Gallagher, J.C.: Evolving dynamical neural networks for adaptive behavior. Adaptive Behavior **1** (1992) 91–122

[10] Nolfi, S., Elman, J.L., Parisi, D.: Learning and evolution in neural networks. Adaptive Behavior **2** (1994) 5–28

[11] Gomez, F.J.: Robust Nonlinear Control through Neuroevolution. PhD thesis, Department of Computer Sciences, The University of Texas at Austin (2003)

[12] Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. Machine Learning **22** (1996) 11–32

[13] Littman, M.L.: Memoryless policies: theorectical limitations and practical results. In: From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior, Cambridge, MA, MIT Press (1994)

[14] Wieland, A.: Evolving neural network controllers for unstable systems. In: Proceedings of the International Joint Conference on Neural Networks (Seattle, WA), Piscataway, NJ: IEEE (1991) 667–673